Abstract Class

- Abstract classes are very similar to interfaces. You can't instantiate either of them. Both types may contain a mix of methods declared with, or without a method block.
- With abstract classes, you can declare fields that aren't static and final, instance fields in other words.
- Also with abstract classes, you can use any of the four access modifiers for its concrete methods.
- You can also use all but the private access modifier, for its abstract methods.
- An abstract class can extend only one parent class, but it can implement multiple interfaces.
- When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class.
- However, if it doesn't, then the subclass must also be declared abstract.

Use an Abstract class when...

You want to share code, among several closely related classes (Animal for example, with fields, name, age...).

You expect classes that extend your abstract class, to have many common methods or fields, or require access modifiers other than public.

You want to declare non-static or non-final fields (for example, name, age), so this enables you to define methods, that can access and modify the state of an object (getName, setName).

You have a requirement for your base class, to provide a default implementation of certain methods, but other methods should be open to being overridden by child classes.

Summary: An abstract class provides a common definition, as a base class, that multiple, derived classes can share.

Interface

- An interface is just the declaration of methods, which you want some classes to have, it's not the implementation.
- In an interface, we define what kind of operation an object can perform. These operations are defined by the classes that implement the interface.
- Interfaces form a contract between the class, and the outside world, and this contract is enforced at build time, by the Java compiler.
- You can't instantiate interfaces, but they may contain a mix of methods declared with, or without an implementation.
- All methods on interfaces, declared without a method body, are automatically public and abstract.
- An interface can extend another interface.

Interface

- Interfaces are more flexible, and can deal with a lot more stress on the design of your program, because they aren't part of the class hierarchy.
- A best practice way of coding, is commonly called Coding to an Interface.
- By introducing interfaces into your program, you're really introducing points of variation, at which you can plug in different implementations for that interface.
- **Summary: The interface decouples the "what", from the "how", and is used to make different types, behave in similar ways.**
- Since Java 8, interfaces can now contain default methods, so in other words methods with implementation. The keyword **`default`** is used mostly for backwards compatibility. Public static methods were also introduced in Java 8.
- Since Java 9, an interface can also contain private methods, commonly used when default methods share common code.

Use an Interface when...

- You expect that unrelated classes will implement your interface. For example, two of Java's own interfaces, Comparable and Cloneable, can be implemented by many unrelated classes.
- You want to specify the behavior of a particular data type, but you're not concerned about who implements its behavior.
- You want to separate different behavior.

Interfaces are the used in many of Java's own features

- I've briefly discussed some interfaces, like List and Queue, and their implementations, **ArrayList** and **LinkedList**. These are part of what Java calls it's **Collections Framework**.
- Interfaces are also the basis for many of the features that are coming up, for example **lambda expressions**, which were introduced in JDK8.
- Another example is **Java's database connectivity support, or JDBC**, built almost entirely with interfaces. The concrete implementation of methods, is different for each database vendor, and comes in the form of JDBC drivers. This enables you to write all database code, without being concerned about the details of the database, you're connected to.

Interface vs. Abstract Class

I've said that interfaces and abstract classes are both abstracted **types, and** abstracted types are used as reference types in code.

The table on this slide is a summary of the similarities and differences.

	Abstract Class	**Interface**
An instance can be created from it	No	No
Has a constructor	Yes	No
Implemented as part of the Class Hierarchy. Uses Inheritance	Yes (in extends clause)	No (in implements clause)
records and enums can extend or implement?	No	Yes
Inherits from java.lang.Object	Yes	No
Can have both abstract methods and concrete methods	Yes	Yes (as of JDK 8)
Abstract methods must include abstract modifier	Yes	No (Implicit)
Supports default modifier for it's methods	No	Yes (as of JDK 8)
Can have instance fields (non-static instance fields)	Yes	No
Can have static fields (class fields)	Yes	Yes - (implicitly **`public static final`**)